Package 'Rserve'

March 23, 2026

Version 1.8-19

Title Versatile R Server

Author Simon Urbanek [aut, cre, cph]
(https://urbanek.org, <https://orcid.org/0000-0003-2297-1732>)

Maintainer Simon Urbanek <Simon.Urbanek@r-project.org>

Depends R (>= 1.5.0)

Suggests RSclient

SystemRequirements libR, GNU make

Description Rserve is a versatile, scalable server enabling the efficient use of R from other applications through variety of protocols including QAP, WebSockets, HTTP and HTTPS. It acts as a server (TCP/IP or local sockets) which allows binary requests to be sent to R. Every connection has a separate workspace and working directory. Client-side implementations are available for many popular languages allowing applications to use facilities of R without the need of linking to the R binary. Rserve supports remote connections, user authentication and file transfer. A simple R client is included in this package as well. It also supports OCAP mode for secure remote procedure calls, including support for full event loop, asynchronous results/graphics and console I/O.

License GPL-2 | file LICENSE

URL https://www.rforge.net/Rserve/

BugReports https://github.com/s-u/Rserve/issues/

Biarch true

NeedsCompilation yes

Contents

ocap 2
Rserve 3
Rserve.eval 4
Rserve.http.add.static 6

run.Rserve 7
self 8
ulog 9

Index **10**

`ocap` *Object Capability (OCAP) Functions*

Description

The following functions are only meaningful when used by code that is run inside Rserve in object-capability (OCAP) mode. See Rserve Wiki for details.

`ocap` registers a function as a capability and returns the reference.

`resolve.ocap` takes a capability reference and returns the function representing the capability.

`Rserve.context` retrieves or sets the current context for out-of-band (OOB) messages (see also `Rserve.eval` for specifying contexts during evaluation).

Usage

```
ocap(fun, name = deparse(substitute(fun)))
resolve.ocap(ocap)
Rserve.context(what)
```

Arguments

`fun`	function to register
`name`	description of the function, only for informational and logging purposes
`ocap`	reference previously obtained by a call to `ocap`
`what`	if present, sets the context to the supplied value. If missing, the function returns the current context

Value

`ocap` returns the new capability reference, it will be an object of the class `"OCref"`.

`resolve.ocap` returns the function corresponding to the reference or `NULL` if the reference does not exist. It will raise an error if `ocap` is not a valid `"OCref"` object.

`Rserve.context` returns the current context

Author(s)

Simon Urbanek

`Rserve` *Server providing R functionality to applications via TCP/IP or local unix sockets*

Description

Starts Rserve in daemon mode (unix only). Any additional parameters not related to Rserve will be passed straight to the underlying R. For configuration, usage and command line parameters please consult the online documentation at http://www.rforge.net/Rserve. Use `R CMD Rserve --help` for a brief help.

The `Rserve` function is provided for convenience only.

On Windows the `Rserve()` function sets up the `PATH` to include the current R.DLL so that Rserve can be run.

Usage

```
# R CMD Rserve [<parameters>]

Rserve(debug = FALSE, port, args = NULL, quote=(length(args) > 1), wait, ...)
```

Arguments

`debug`	determines whether regular Rserve or debug version of Rserve (`Rserve.dbg`) should be started.
`port`	port used by Rserve to listen for connections. If not specified, it will be taken from the configuration file (if present) or default to 6311
`args`	further arguments passed to Rserve (as a string that will be passed to the `system` command - see `quote` below).
`quote`	logical, if `TRUE` then arguments are quoted, otherwise they are just joined with spaces
`wait`	wait argument for the `system` call. It defaults to `FALSE` on Windows and `TRUE` elsewhere.
`...`	other arguments to be passes to `system`.

Details

Rserve is not just a package, but an application. It is provided as a R package for convenience only. For details see http://www.rforge.net/Rserve

Note

`R CMD Rserve` will only work on unix when installed from *sources* and with sufficient permissions to have write-rights in `$R_HOME/bin`. Binary installations have no way to write in `$R_HOME/bin` and thus `Rserve()` function described above is the only reliable way to start `Rserve` in that case.

Java developers may want to see the `StartRserve` class in `java/Rserve/test` examples for easy way to start `Rserve` from Java.

Rserve can be compiled with TLS/SSL support based on OpenSSL. Therefore the following statements may be true if Rserve binaries are shipped together with OpenSSL: This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/).

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com). They are not true otherwise.

Author(s)

Simon Urbanek

See Also

`run.Rserve`

`Rserve.eval` *Evaluate expressions in a REPL-like fashion*

Description

`Rserve.eval` evaluates a given expression in a way that is very close to the behavior on the console Read/Evaluate/Print Loop (REPL). Among other things this means printing the result of each expression if visible. The function is guaranteed to not raise an error and in case of an error it returns an object of class `Rserve-eval-error` with details including the error and the stack trace.

Usage

```
Rserve.eval(what, where = .GlobalEnv, last.value = FALSE, exp.value = FALSE,
            context = NULL, handlers = list(error=.save.condition))
```

Arguments

`what`	expressions to evaluate
`where`	environment to evaluate in
`last.value`	logical, if `TRUE` then the result of the evaluation is returned, otherwise the evaluation is only performed for its side-efects and returns `TRUE` instead.
`exp.value`	logical, it `TRUE` then an error object will include the actual expression that triggered the error, otherwise it will only store the index of the expression in `what`.
`context`	optional object that will be used as the Rserve context for the duration of the evaluation (see `Rserve.context`).
`handlers`	optional named list of calling handlers to register for the duration of the evaluation. The default is to register an `error` handlers which stores the error condition so it can be reported in the result - see below.

Details

If `what` contains one or more expressions, they are evaluated one by one while printing the result of each if visible. Upon error subsequent expressions are not evaluated. If `what` is not an expression then the only a single evaluation of `what` is performed and the result is not printed.

The main purpose of this function is to implement console front-ends where the front-end uses `parse()` + `Rserve.eval()` to simulate the action of a GUI. Because the function returns in all circumstances it allows clients to rely on a well-define messaging behavior.

Value

If the evaluation triggered an error, the result is an object of class `Rserve-eval-error` with components:

`error`	character, error message
`traceback`	list of contexts in the traceback
`expression`	if `what` contains multiple expressions then this will be either an index to the expression that caused the error (`exp.value=FALSE`) or the actual expression (otherwise).
`context`	current Rserve context, `NULL` if none has been set
`condition`	if any condition has been saved via `.save.condition` (which is the default) then on error the captured condition object is stored here, `NULL` otherwise

If the evaluation finished without an error then the result is either `TRUE` if `last.value=FALSE` or the value of the last expression otherwise.

Note

Rserve versions up to 1.8-10 did not include the `condition` component, no calling handlers were registered and there was no `condition` component in the result. To replicate that behavior or if you don't need that information, you can set `handlers=NULL` which removes the overhead of adding calling handlers.

No error checking is performed on the `handlers` parameter, so make sure it is avalid, named list of functions, otherwise an error will occur at evaluation time.

Author(s)

Simon Urbanek

Examples

```
g <- function() stop("foo")
f <- function() g()
(Rserve.eval(expression(f())))
(Rserve.eval(parse(text="1:5\n1+1")))
(Rserve.eval(quote(1+1), last.value=TRUE))

error_with_condition = function(object = NULL) {
  cond = errorCondition("this is a custom error with condition",
                        object = object,
                        class = "CustomError")
  stop(cond)
}
str(Rserve.eval(quote(error_with_condition("hello")), last.value = TRUE))
```

`Rserve.http.add.static` *Add static file handler to HTTP/HTTPS server*

Description

`Rserve.http.add.static` installs a new static handler to be used by the HTTP/HTTPS servers. It will trigger only if the path prefix matches `prefix` and will map the subsequent portion of the path in the request URL to a file system location specified by `path`. If the resulting item in the file system is a directory, then `index` (if set) will be appended to the path and served instead (if it is a file).

`Rserve.http.rm.all.statics` removes all static handlers from the current R session.

Usage

```
Rserve.http.add.static(prefix, path, index = NULL, last = FALSE)
Rserve.http.rm.all.statics()
```

Arguments

`prefix`	string, path prefix for which this handler will be used
`path`	string, path in the filesystem used as root to serve the content
`index`	optional string, will be appended to the file system path if the target is a directory (typical value is `"index.html"`).
`last`	logical, if `FALSE` then processing continues to other handlers if the target does not exist. If `TRUE` then all requests for the prefix will be handled only by this handler, possible resulting in "404 not found" result if the reqeusted file does not exist.

Details

The HTTP/HTTPS server supports both static and dynamic handlers. The typical use is to invoke `.http.request` function in R for dynamic handling, but it also supports static maps of URL paths to file system locations. The static handlers are checked first.

`Rserve.http.add.static` installs a new static handler, adding it to the list of handlers. The handlers are consulted in the order that they are added.

The static handler supports conditional GETs and relies on the file system modification times to determine if a file has been modified.

Value

The return value is considered experimental and may change in the future: Integer, number of active handlers (which is the same as the index of this handler).

Author(s)

Simon Urbanek

See Also

run.Rserve

Examples

```
## standard handler serving all files in the current working directory
## and consults index.html in directories if no file is specified.
Rserve.http.add.static("/", getwd(), "index.html", TRUE)

## start the server with:
##   run.Rserve(http.port=8080, qap=FALSE)
```

---

run.Rserve *Start Rserve within the current R process.*

Description

`run.Rserve` makes the current R process into an Rserve instance. Rserve takes over until it is shut down or receives a user interrupt signal. The main difference between `Rserve` and `run.Rserve` is that `Rserve` starts a new process, whereas `run.Rserve` turns the current R session into Rserve. This is only possible if there are no UI elements or other parts that could interfere with the prepation of `Rserve`.

`stop.Rserve` stops currently running background servers. This only applies to servers started using `background=TRUE`.

Usage

```
run.Rserve(..., config.file = "/etc/Rserve.conf", background = FALSE)
stop.Rserve()
```

Arguments

`...`	all named arguments are treated as entries that would be otherwise present in the configuration file. So argument `foo="bar"` has the same meaning as `foo bar` in the configuration file. The only exception is that logical values can be used instead of `enable`/`disable`. Some settings such as `uid` are not relevant and thus ignored.
`config.file`	path of the configuration file to load in the Rserve. It will be loaded before the above settings and is optional, i.e. if the file is not present or readable it will be ignored.
`background`	logical, the default `FALSE` starts the server and does not return until all servers have been shut down - typically in response to an interrupt. If this argument is set to `TRUE` then the server is started in teh background of this R session and control is returned immediately (currently not supported on Windows). In that case requests will be only processed if no other computation is running in R, but the R console can be still used to modify the session. Such background servers can be stopped with the `stop.Rserve` function.

Value

Returns `TRUE` after the Rserve was shut down.

Author(s)

Simon Urbanek

See Also

Rserve

---

self *Functions usable for R code run inside Rserve*

Description

The following functions can only be used inside Rserve, they cannot be used in stand-alone R. They interact with special features of Rserve. All commands below will succeed only if Rserve has been started with `r-control enable` configuration setting for security reasons.

`self.ctrlEval` issues a control command to the Rserve parent instance that evaluates the given expression in the server. The expression is only queued for evaluation which will happen asynchronously in the server (see `RSserverEval` in `RSclient` package for details). Note that the current session is unaffected by the command.

`self.ctrlSource` issues a control command to the Rserve parent instance to source the given file in the server, see `RSserverSource` in the `RSclient` package for details.

`self.oobSend` sends a out-of-band (OOB) message with the encoded content of `what` to the client connected to this session. The OOB facility must be enabled in the Rserve configuration (using `oob enable`) and the client must support OOB messages for this to be meaningful. This facility is not used by Rserve itself, it is offered to specialized applications (e.g. `Cairo` supports asynchronous notification of web clients using WebSockets-QAP1 tunnel to dynamically update graphics on the web during evaluation).

`self.oobMessage` is like `self.oobSend` except that it waits for a response and returns the response.

Usage

```
self.ctrlEval(expr)
self.ctrlSource(file)
self.oobSend(what, code = 0L)
self.oobMessage(what, code = 0L)
```

Arguments

`expr`	R expression to evaluate remotely
`file`	path to a file that will be sourced into the main instance
`what`	object to include as the payload fo the message
`code`	user-defined message code that will be ORed with the `OOB_SEND`/`OOB_MSG` message code

Value

`oobMessage` returns data contained in the response message.

All other functions return `TRUE` (invisibly).

Author(s)

Simon Urbanek

Examples

```
## Not run:
  self.ctrlEval("a <- rnorm(10)")
  self.oobSend(list("url","http://foo/bar"))

## End(Not run)
```

`ulog` *Micro Logging*

Description

`ulog` logs the supplied message using the `ulog` facility which typically corresponsed to syslog. See `ulog` Rserve configuration for the various endpoints supported by ulog (local, UDP/TCP remote, ...).

This function is guaranteed to be silent regardless of the ulog setting and is intended to have minimal performance impact.

Note: if Rserve is compiled with `-DULOG_STDERR` (also implied in the debug build) then ulog messages are also emitted on `stderr` with `"ULOG: "` prefix.

Please note that this `ulog` function is governed by the Rserve settings, and NOT the `ulog` package settings. The latter is a general port of the `ulog` logging facility to R, while `Rserve::ulog` is specific to to the Rserve process.

Usage

```
ulog(...)
```

Arguments

`...`	message to log

Value

The logged string constructed from the message, invisibly

Author(s)

Simon Urbanek

Examples

```
ulog("INFO: My application started")
```

Index

∗ **interface**
    ocap, 2
    Rserve, 3
    Rserve.http.add.static, 6
    run.Rserve, 7
    self, 8
    ulog, 9
∗ **manip**
    Rserve.eval, 4

ocap, 2

resolve.ocap *(ocap)*, 2
Rserve, 3, *7*, *8*
Rserve.context, *4*
Rserve.context *(ocap)*, 2
Rserve.eval, *2*, 4
Rserve.http.add.static, 6
Rserve.http.rm.all.statics
        (Rserve.http.add.static), 6
run.Rserve, *4*, *7*, 7

self, 8
stop.Rserve *(run.Rserve)*, 7
system, *3*

ulog, 9